**UNIVERSIDADE FEDERAL DE SERGIPE**
**CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS**
**PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO**

# Identificação e análise de clones de códigos heterogêneos em um ambiente corporativo de desenvolvimento de *software*

**JOSÉ JORGE BARRETO TORRES**

**SÃO CRISTÓVÃO - SE**

**Agosto, 2016**

**UNIVERSIDADE FEDERAL DE SERGIPE**
**CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS**
**PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO**

**JOSÉ JORGE BARRETO TORRES**

# Identificação e Análise de Clones de Códigos Heterogêneos em um Ambiente Corporativo de Desenvolvimento de *Software*

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Ciência da computação (PROCC) da Universidade Federal de Sergipe (UFS) como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Methanias C. R. Junior

**SÃO CRISTÓVÃO - SE**

**Agosto, 2016**

**JOSÉ JORGE BARRETO TORRES**

# Identificação e Análise de Clones de Códigos Heterogêneos em um Ambiente Corporativo de Desenvolvimento de *Software*

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Ciência da computação (PROCC) da Universidade Federal de Sergipe (UFS) como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação.

## BANCA EXAMINADORA

Prof. Dr. Methanias Colaço Júnior, Presidente
Universidade Federal de Sergipe (UFS)

Prof. Dr. Michel dos Santos Soares
Universidade Federal de Sergipe (UFS)

Prof. Dr. Glauco de Figueiredo Carneiro
Universidade Salvador (UNIFACS)

**Identificação e Análise de Clones de Códigos Heterogêneos em um Ambiente Corporativo de Desenvolvimento de *Software***

Este exemplar corresponde à redação parcial da Dissertação de Mestrado, sendo o Exame de Defesa do mestrando **JOSÉ JORGE BARRETO TORRES** para ser aprovado pela Banca Examinadora.

São Cristóvão – SE, 31 de AGOSTO de 2016.

_____

Prof. Dr. Methanias Colaço Júnior
Orientador

_____

Prof. Dr. Michel dos Santos Soares
Membro

_____

Prof. Dr. Glauco de Figueiredo Carneiro
Membro

# RESUMO

A exigência por acelerar o desenvolvimento de software nas empresas desencadeia uma série de problemas relacionados à organização do código. As equipes de desenvolvimento, pressionadas a cumprir prazos ditados pela área de negócio, adotam a prática ruim de copiar e colar código. Assim, os clones são criados e povoam os repositórios de software dessas companhias, tornando o aprimoramento e manutenção dos sistemas cada vez mais dificultado. Linguagens de programação que possuem características do paradigma de orientação a objetos tendem a facilitar ainda mais o processo de abstração de código e de reaproveitamento. No entanto, uma questão pode ser feita: uma mesma equipe, trabalhando com diversos tipos de linguagens, sofre influência destes tipos, no que diz respeito à diminuição da incidência de clones? Este trabalho propôs uma abordagem para identificar, analisar e comparar clones em repositórios heterogêneos de software, com uma análise tênue do perfil da equipe envolvida. A avaliação experimental da abordagem foi realizada por meio de dois experimentos controlados, os quais visaram a detecção e a avaliação de clones, utilizando e adaptando o ferramental disponível no mercado. Esta avaliação foi executada *in-vivo,* em um ambiente organizacional real, o qual possuía uma grande quantidade de aplicações e linhas de código fechado disponíveis para análise. Os resultados finais não apresentaram relação direta com a quantidade de linhas de código das aplicações. Sistemas de linguagem procedural apresentaram menor incidência de clones e, no conflito entre sistemas de código aberto e fechado, ambos tiveram resultados similares no que diz respeito à manifestação de clones de código-fonte.

**Palavras-chave:** Mineração de Repositórios de Software; Clones; Engenharia de Software Experimental; Código Fechado de Ambientes Corporativos.

# ABSTRACT

The demand for speeding up software development inside corporations triggers a series of issues related to coding organization. Software development teams have to achieve business deadlines, so they adopt the bad practice to copy-and-paste code. In this way, clones populate software repositories and hinder the improvement or maintenance of systems. Programming languages with object-oriented paradigm characteristics tend to make easy coding abstraction and reuse processes. However, a question arises: the same team working with several kinds of programming languages are influenced by their paradigms regarding the decrease of cloning incidence? This work proposed an approach to identify, analyze and compare clones inside heterogeneous software repositories without consider the development team profile. The experimental evaluation of the approach was possible thru two controlled experiments which aimed to detect and evaluate clones, using and adapting tools available on market. This evaluation was executed inside an organizational environment, which owned several applications with closed-source code but available to analysis. The final results showed no relationship to the amount of application code lines. Procedural language systems had a lower clone incidence and, when conflicting open and closed source systems, both had similar results regarding to the manifestation of source-code clones.

**Keywords:** Mining Software Repositories; Clones; Experimental Software Engineering; Closed-source projects.

SUMÁRIO

# 1. Introdução

Ao introduzir este trabalho, pretende-se descrever a conjuntura do cenário geral das equipes de desenvolvimento de software de empresas privadas com relação aos clones de código, apresentando o problema chave desta pesquisa e o argumento para realização da mesma.

## 1.1. Contextualização

A exigência por acelerar o desenvolvimento de *software* aliada à ausência de padrões e à inexistência de políticas internas que implementam melhores práticas, desencadeiam uma série de problemas relacionados à organização do código. As equipes de desenvolvimento, pressionadas por cumprir prazos ditados pela área de negócio, adotam a prática ruim de copiar e colar código. Dessa maneira, os clones povoam os repositórios de *software* das organizações, dificultando o aprimoramento ou manutenção dos programas (BAXTER *et.al.,* 1998).

Baxter *et.al.* (1998) descreve alguns motivos para a existência de clones:

- Reutilização de código: Grande parte do código presente em sistemas legados é produzido por reutilização de código existente. Programadores que desejam implementar uma nova funcionalidade encontram algum trecho de código semelhante ao desejado, efetuando uma cópia e posterior modificação;
- Estilos de codificação: Alguns fragmentos de código utilizados em mensagens padronizadas de retorno ao usuário são copiados para manter um padrão.
- Instâncias de computações parecidas: Códigos que efetuam computações semelhantes também são frequentemente clonados, mesmo sem o ato de copiar e colar, pelo fato das operações serem parecidas;
- Falhas ao utilizar os tipos abstratos de dados: Existem clones que são resultado da duplicação de instruções idênticas que trabalham com tipos de dados diferentes.
- Melhoria de desempenho: Sistemas que possuem restrições de tempo e precisam de frequentes otimizações para replicação de

cálculos, especialmente quando o compilador não fornece o suporte à inserção de expressões em linha;

- Clones acidentais: São trechos de código similares espalhados pelo repositório onde, em princípio, não possuem qualquer ligação funcional. À medida que as aplicações aumentam de tamanho, esse tipo de acidente ocorre com mais frequência.

Linguagens de programação que possuem características que seguem o paradigma de orientação a objetos são acompanhadas por uma proposta de utilização mais intensa dos recursos de abstração e reaproveitamento de código. Essa proposta também existe no paradigma procedural, que fornece os procedimentos e funções para simplificação e aproveitamento no desenvolvimento das aplicações. O reaproveitamento de código-fonte em sistemas orientados a objetos ocorre por meio de diferentes mecanismos, como herança, bibliotecas compartilhadas, entre outros. Apesar de algumas abordagens de design de sistemas facilitarem a reutilização de código, componentes de *software* que não foram construídos pensando-se em reutilização precisam ser aprimorados durante alguma expansão ou mudança de requisitos. (TORRES, JUNIOR e SANTOS, 2016)

O estudo de Kapser e Godfrey (2008) aborda a existência de padrões para clonagem durante seus estudos de caso, além de discutirem as vantagens e desvantagens associadas ao uso destes clones em termos de desenvolvimento e manutenção. Através de toda observação proveniente do estudo, foi notado que nem sempre a refatoração é a melhor solução para todos os onze padrões de clonagem descobertos. Kim, Sazawal e Notkin (2005) também sugerem que, com o uso apropriado de padrões de projeto, pode-se reduzir a incidência de clones nos repositórios de código-fonte.

Baxter *et.al.* (1998 *apud* BAKER, 1995 e LAGUE, 1997) expõe dados obtidos a partir de trabalhos anteriores que indicam que cerca de 5 a 10% do código-fonte de programas de computadores é duplicado. Reforça também que, programadores seguem copiando trechos de código semelhantes às necessidades do momento, efetuando alguma customização nesse código para adaptá-lo a um novo contexto. Essa atitude revela a intenção de implementar algum tipo de abstração. A

ação de copiar e colar código ainda quebra um princípio da engenharia de software: o encapsulamento.

Autores como Sarala e Deepika (2013) também inferem outro problema: todos os fragmentos de código similares podem conter um mesmo *bug*, se o trecho de origem apresenta o mesmo problema.

### 1.2. Análise do Problema

Alguns trabalhos já exploram a problemática dos clones de *software*, fornecendo soluções remediadoras por meio de ferramentas que utilizam variadas técnicas de detecção e fatoração, seja por meio de Grafos de Dependência (HIGO e KUSUMOTO, 2011) ou Árvores de Abstração de Sintaxe (BAXTER *et.al.*, 1998). Rehman *et.al.*(2012) propõe outra técnica que utiliza *arrays* de duas dimensões e promete ser eficiente na análise de múltiplas linguagens de programação. Rattan, Rajesh e Maninder (2013) apresentam uma extensa revisão sistemática e destacam os benefícios da gestão de clones de software, identificando a necessidade do desenvolvimento de técnicas de detecção de clones semânticos. Kim e Notkin (2009) estudaram mudanças sistemáticas de estrutura de código efetuadas por engenheiros de software profissionais de uma grande empresa de *e-commerce*, com o objetivo de identificar anomalias e melhorar a detecção de inconsistências, que inclui também a incidência da programação por meio de clones.

A facilidade de programar uma nova funcionalidade simplesmente copiando algo similar existente e modificando apenas as partes necessárias é atrativa. A pressão para entregar os trabalhos no curto prazo também contribui para o acontecimento da duplicação de código. A atitude de "copiar-e-colar" para implantar algo semelhante já sugere algum tipo de abstração, que poderia ser desenvolvida de modo mais eficiente para aproveitar a reutilização (BAXTER *et.al.*, 1998).

Na mesma linha de questionamento, a construção de drivers de dispositivos também gera uma grande quantidade de similaridades entre códigos, já que grande parte desse tipo de programa voltado a uma mesma plataforma é praticamente idêntico, sendo modificados apenas alguns atributos e parâmetros (MA e WOO, 2007).

Khatoon e Mahmood (2011) ressaltam que, apesar da maneira "copiar-e-colar" ser mais produtiva, essa atitude por parte dos programadores pode causar um grave problema de manutenção, por exemplo, em caso de bugs. Se um bug foi encontrado em uma parte de código que foi clonado em vários outros pedaços, todos esses clones devem ser corrigidos para que o bug seja solucionado por completo.

Outro motivo para a existência dos clones é o que Marcus e Maletic (2001) chamam de "a reinvenção da roda". Isso acontece quando o desenvolvedor desconhece a solução de um problema programável existente em sua biblioteca de códigos e resolve criar esse recurso do zero. Dessa situação, surgem clones mais difíceis de detectar, também conhecidos como "*wide miss*" clones. Neste contexto, cópias que envolvem Tipos Abstratos de Dados (*TAD*) são denominadas clones conceituais de alto nível.

A problemática deste projeto é analisar a eficiência de diferentes paradigmas com relação à incidência de clones, utilizando para isso o ferramental disponível no mercado. Essa verificação é realizada em repositórios de código-fonte de uma organização particular.

Assim, os questionamentos que rodeiam esse problema são:

a. As linguagens com características do paradigma de orientação a objetos são mais eficientes do que as procedurais, com relação ao surgimento de clones?
b. Em repositórios de código aberto, existe uma tendência maior de aparecimento de clones?

A partir dessas questões, foram formuladas as seguintes hipóteses:

- Hipótese $H0^P$: O paradigma não exerce influência no aparecimento de clones.
- Hipótese $H1^P$: Repositórios de linguagens com características do paradigma OO apresentam menos clones de código do que as procedurais.

- Hipótese $H0^{OP}$: Repositórios de organizações privadas têm a mesma incidência de clones de repositórios *Open Source.*
- Hipótese $H1^{OP}$: Repositórios de organizações privadas têm uma incidência menor de clones do que repositórios *Open Source.*

Como será visto ao longo desta dissertação, os resultados obtidos a partir da análise dos repositórios de código fonte explorados não apresentaram evidências de vantagem ao utilizar o paradigma OO, no tocante à manifestação de clones. Em paralelo, os repositórios de organizações privadas apresentaram resultados semelhantes aos *Open Source,* com relação à incidência de fragmentos de código similares.

## 1.3. Justificativa

Este projeto busca identificar e analisar algumas razões da manifestação de clones nos diversos tipos de repositórios de código. A análise e verificação desses repositórios irão contribuir para testar teorias relacionadas ao aparecimento e multiplicação dos clones de *software*. Algumas dessas razões se referem a dúvidas relacionadas ao paradigma das linguagens de programação, experiência dos desenvolvedores e influências do movimento *Open Source versus* códigos fechados pertencentes a empresas privadas.

## 1.4. Objetivos

### Objetivo Geral

Utilizar e analisar técnicas de detecção de clones de códigos heterogêneos em um ambiente corporativo de desenvolvimento de software, identificando evidências experimentais que podem contribuir para disseminação e eliminação de trechos clonados.

### Objetivos Específicos

Os objetivos que dão direcionamento ao cerne desta dissertação são descritos abaixo:

- Adaptação de ferramentas de detecção de clones ao contexto exigido;
- Estudo de caso para análise de ferramentas de detecção de clones em um ambiente corporativo;
- Planejamento e execução de um experimento controlado, confrontando a incidência de clones entre sistemas de código aberto e sistemas privados;
- Planejamento e execução de um experimento controlado, pondo em conflito a manifestação dos clones de código-fonte entre diferentes paradigmas de linguagem de programação.

## 1.5. Metodologia

A metodologia utilizada para o presente estudo consistiu-se de uma pesquisa exploratória, prática e um processo experimental.

A pesquisa exploratória (SEVERINO, 2008) é aquela que busca identificar o estado da arte para a temática proposta, por meio de livros, artigos e ferramentas. Com os resultados dessa pesquisa, foi realizada uma revisão da literatura, que aproveitou os conteúdos que fundamentaram a confecção dos trabalhos. A prática é caracterizada pelo uso de ferramentas que analisaram dados de um ambiente real.

O estudo também passa por um processo experimental, seguindo as diretrizes de Wohlin *et al.* (2012). O experimento é um tipo de pesquisa científica na qual o pesquisador identifica variáveis dependentes e as observa através da manipulação de variáveis independentes. Foi realizada a seleção de repositórios de código-fonte para análise da detecção de clones em um ambiente controlado (*in Vivo*) e, por fim, a comparação da manifestação desses clones entre os repositórios, considerando os paradigmas de programação e o tipo de ambiente de desenvolvimento.

As buscas por clones nos repositórios foram executadas através das mesmas ferramentas utilizadas nos trabalhos relacionados, para garantia de maior confiabilidade nos resultados finais. Todas as

características experimentais são detalhadas nos projetos disponibilizados nos capítulos 3 e 4.

## 1.6. Organização

Este documento está organizado em 5 capítulos que fornecem uma base conceitual e experimental para o seu entendimento. Os tópicos a seguir descrevem o conteúdo de cada um dos capítulos:

- O Capítulo 1 apresenta esta Introdução, explicando as justificativas juntamente com as hipóteses levantadas;
- O Capítulo 2 traz um breve Referencial teórico acerca da temática abordada;
- No Capítulo 3, é disponibilizado um artigo que foi apresentado no ITNG 2016 e publicado em livro pela Springer;
- O Capítulo 4 traz um artigo submetido ao SEKE 2016 e ao SBES 2016, com as devidas correções e recomendações;
- Finalmente, no capítulo 5, é apresentado um compilado de conclusões, contribuições e sugestões de trabalhos futuros.

## 2. Clones: Conceitos e Abordagens

Este capítulo efetua uma trajetória pelos principais conceitos acerca do tema proposto, procurando embasamento para sustentar o trabalho. Aqui são definidos os Clones e suas principais técnicas de detecção.

Um fragmento de programa idêntico a outro – essa é a base da teoria apontada por Baxter *et.al.* (1998), que dá o nome de idioma ao pedaço de código que implementa um conceito de programação. Outra definição é chamada de *near miss* clone, que não representa uma réplica exata do código, mas é considerado um clone por possuir um mesmo resultado semântico. Um *near miss* clone muitas vezes é um pedaço de programa que foi reaproveitado e teve algumas partes internas modificadas, como exemplifica a Figura 1.

```
---------------------------------- CLONE ----------------------------------
Similarity = 0.929411764705882
From 13407 To 13423

Db_err error;
Bool fail;
if ( db_get_int ( DB_CF_VAC_GROUP_REGEN_IV , &  error ) )
{
  vac_regen_group_ob ( );
  return ( 0 );
}
db_put_ts ( DB_VAC_OI_P5_CRYO_REGEN_IN_PROG_TSV , TRUE , & error );
db_put_int ( DB_VAC_CRYO_P5_STATE_IV , VAC_REGEN, & error );
CALL ( fac_n2 ( ) );
alm_activate ( ALM_VAC_P5_REGENING, ALM_DONT_BLOCK , ALM_DONT_ACK ,60 , MDP_NULL ,
MDP_NULL );
---------------------------------------------------------------------------
From 13208 To 13224

Db_err error;
Bool fail;
if ( db_get_int ( DB_CF_VAC_GROUP_REGEN_IV,& error ) )
{
  vac_regen_group_ob ( );
  return ( 0 );
}
db_put_ts ( DB_VAC_OI_P4_CRYO_REGEN_IN_PROG_TSV , TRUE , & error );
db_put_int ( DB_VAC_CRYO_P4_STATE_IV , VAC_REGEN, & error );
CALL ( fac_n2 ( ) );
alm_activate ( ALM_VAC_P4_REGENING,ALM_DONT_BLOCK , ALM_DONT_ACK ,60 , MDP_NULL ,
MDP_NULL );
```

**Figura 1**: Exemplo de detecção de clone. Adaptada de Baxter *et.al*. (1998).

A Figura 1 mostra a detecção de dois blocos de código que diferem apenas pelos nomes dos parâmetros. Isso denota claramente a possibilidade de algum tipo de abstração.

Tempero (2013) define terminologias que auxiliam no entendimento dos clones. A primeira delas é o "fragmento de código", que se traduz em uma sequência de linhas de código de qualquer granularidade, podendo ser a completa definição de um método ou um bloco. Um "par de clones" é definido por possuir dois fragmentos de código parecidos a partir de uma determinada exigência de similaridade. Quando mais de dois fragmentos semelhantes são encontrados, esses são chamados de "*cluster* de clones".

O primeiro passo para a detecção de clones é a transformação do código-fonte em uma Árvore de Abstração de Sintaxe (AST). Após isso, de três algoritmos são aplicados para encontrar os clones. O primeiro, chamado de algoritmo básico, propõe detectar clones da sub-árvore. O segundo, denominado algoritmo de detecção de sequência, concentra-se em encontrar sequências de tamanho variável de clones de sub-árvores e é utilizado essencialmente para detectar clones sequências de declarações e sentenças. O terceiro algoritmo visa buscar por clones mais complexos, generalizando combinações com outros clones. O trabalho de Baxter *et.al.*(1998), não executa a remoção ou refatoração dos clones. Ao invés disso, sugere uma função que abstraia o que foi implementado.

Schwarz, Lungu e Robbes (2012), definem os clones através das seguintes notações:

- Tipo 1: Clones de códigos idênticos
- Tipo 2: Clones que possuem mesma estrutura, porém, nomes de identificadores diferentes;
- Tipo 3: Clones que possuem mais modificações na estrutura.

Os Tipos 1 e 2 são detectados com maior facilidade através de técnicas que utilizam *Hash* e *Bad Hash* (também explorada pelo autor anterior). O Tipo 3 requer um pouco mais de especialização dessas técnicas, para que a detecção seja mais confiável.

Pesquisadores, a exemplo de Balazinska *et.al.* (1999), investem em projetos que exploram a utilização de mecanismos de detecção de clones para aplicar na fatoração de códigos-fonte. Por meio de ferramentas que detectam as partes de código clonadas, são executadas ou sugeridas modificações que utilizam reaproveitamento de código através de herança, bibliotecas compartilhadas ou outros recursos suportados pela linguagem analisada. No artigo referido, algumas técnicas de detecção de clones são analisadas. Algumas delas, baseadas na leitura completa do código-fonte e identificação de duplicações exatas utilizando impressões digitais (*fingerprints*). Algumas duplicações são quase idênticas, diferindo apenas nos nomes das variáveis e constantes. Outras abordagens focam em capturar sequências de instruções (blocos *BEGIN-END* ou funções) e permitem a detecção de blocos similares através de métricas relacionadas a aspectos de sequências de instruções, variáveis definidas, *layout*, etc. São encontradas também técnicas de comparação de padrões como a "correspondência dinâmica programável".

Sarala e Deepika (2013) descrevem um processo de refatoração de código em C#, auxiliado pela detecção de clones. O processo visa melhorar a qualidade e reduzir a complexidade de um software. A pesquisa desses autores, realiza a detecção dos clones através de sua semântica, com o adcional de implantar a refatoração através de um novo algoritmo de "Refatoração Gráfica Abstrata Semântica".

Ekoko e Robillard (2007) constroem uma ferramenta para alertar o programador a respeito de clones, durante a construção do sistema. A técnica proposta conta com uma representação heurística de regiões de clones que identifica localizações de código-fonte com uma série de métodos, utilizando uma combinação de informações léxicas, sintáticas e estruturais. Assim, antecipa e propicia a correção em tempo de desenvolvimento, antes mesmo que a aplicação entre em produção. Essa representação abstrata é chamada de Descritor de Regiões de Clones (DRC) e suporta o rastreamento de clones em diferentes versões de um sistema de software. A ferramenta desenvolvida para essa busca é denominada de CloneTracker. Ela recebe como entrada, uma saída de uma ferramenta de detecção de clones e automaticamente produz os

DRCs que representa as regiões para diferentes grupos de clones. A partir disso, rastreia automaticamente enquanto o código evolui, alertando os desenvolvedores sobre modificações na região clonada e suportando múltipla edição de regiões de clones.

Métodos básicos anteriormente mencionados também são combinados com a técnica de similaridade semântica para detecção de clones conceituais de alto-nível. Esses clones são implementações de Tipos Abstratos de Dados (TAD) que possuem mesmo valor semântico. Aplicados em funções, arquivos e segmentos de código, esses métodos melhoram a qualidade da detecção de clones através da computação de similaridade entre elementos de software baseados em informações estruturais. Para automatizar completamente esse processo é necessário combiná-lo com outros métodos de detecção de clones. (MARCUS e MALETIC, 2001):

Ma e Woo (2007) trabalham em uma proposta de detecção de clones em drivers de dispositivos. São conceituadas definições de clones intra-código, as que estão dentro de um mesmo arquivo-fonte e extra-código, para clones detectados em arquivos diferentes. No projeto de teste, é feito uso de uma ferramenta chamada *CCFinder* que propicia a detecção de códigos clone por métricas variadas. As mais utilizadas pelos autores são:

- NBR(f): a quantidade de arquivos fonte que incluem um ou mais fragmentos de código relacionados com os clones de intra-código do arquivo f.
- RSA(f): o percentual de tokens do arquivo f cobertos por clones intra-código.

O valor NBR é representado por um inteiro maior que zero e, quanto maior esse valor, maior é a quantidade de arquivos similares. A métrica RSA representa o percentual de similaridade entre esses arquivos. Se um arquivo possui um valor de RSA próximo a 100%, significa que provavelmente esse arquivo foi criado por meio de uma cópia.

A técnica de detecção baseada em grafos de dependência – conhecida como *Program Dependency Graph* (PDG) – é explorada por

Higo e Kusumoto (2011), onde métodos heurísticos são utilizados para aprimorar o consumo de tempo que essa metodologia leva para detectar clones. Em outro trabalho publicado por Higo *et.al.* (2011) a detecção é implementada através do PDG com uma abordagem diferente, de maneira incremental.

Uma proposta de detecção de clones desenvolvida por Rehman *et.al.* (2012), é uma técnica que utiliza *arrays* de duas dimensões e promete ser rápida e eficiente na análise de múltiplas linguagens de programação. A sequência básica da detecção resume-se em: a) Ler o código-fonte; b) Gerar *Token*: converter o código em *tokens* e inseri-los em um *dataset* de duas dimensões; c) Configurar um valor de *hash* para cada *token* convertido; d) Detectar os clones a partir dos valores de *hash* gerados.

Para detecção de clones em bibliotecas de construção de código, Ishihara *et.al.* (2012) propõe uma detecção mais apropriada com um nível de granularidade baseado em métodos. A técnica é aplicada em um enorme *data set*, contendo 360 milhões de linhas de código distribuídas em 13 mil projetos.

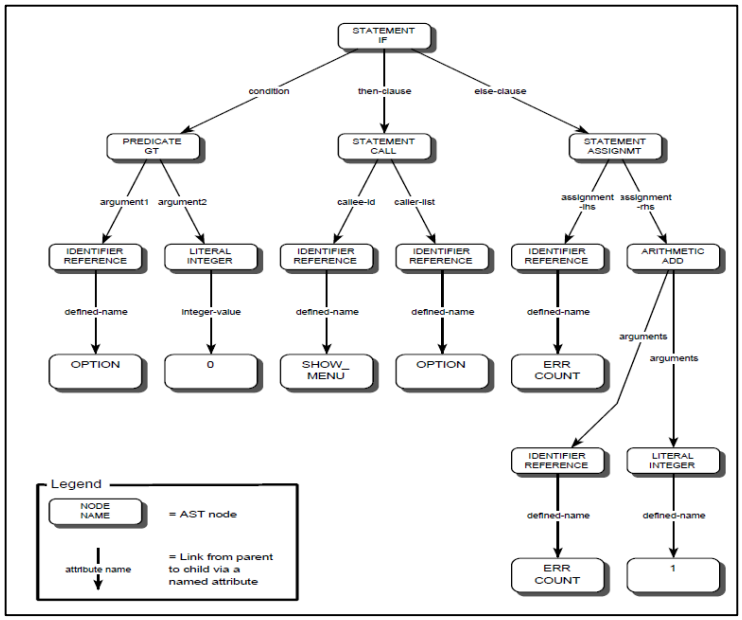


**Figura 2**: Um exemplo de Árvore Sintática Abstrata. Fonte: Balazinska *et.al.* (1999).

Técnicas de detecção de clones utilizadas pela maior parte dos trabalhos consultados utilizam as Ávores de Abstração de Sintaxe, primeiramente explorada por Baxter *et.al.*(1998), exibida através de um exemplo na Figura 2. É possível notar, ainda na Figura 2, as abstrações de cada parte de um bloco de código, fazendo parte da estrutura hierárquica da árvore *AST*.

## 3. Artigo 1 (ITNG)

### Mining Source Code Clones in a Corporate Environment

Jose J. Torres[1], Methanias C. Junior[1], Francisco R. Santos[2]

[1]Federal University of Sergipe – UFS, Sao Cristovao, Sergipe, Brazil.
jorgesamango@gmail.com, mjrse@hotmail.com
[2]Federal Institute of Sergipe – IFS, Lagarto, Sergipe, Brazil.
frchico@gmail.com

***Abstract.*** Many researches around code clone detection rely on Open Source Software (OSS) repositories to execute their studies. These cases do not reflect the corporative code development scenario. Big Companies repositories' are protected from the public's access, so their content and behavior remain as a black box on the researchers' viewpoint. This article presents an experiment performed on systems developed in a large private education company, to observe and compare the incidence of cloned code on proprietary software with other studies involving open source systems, using different similarity thresholds. The results indicate that the closed-source repository presents similar clone incidence as the OSS ones.

**Keywords:** Proprietary Software; Mining Software Repositories; Clones; Experimental Software Engineering; Closed-source projects.

### Introduction

The demand for speeding up software development allied to the lack of patterns and the inexistence of internal policies to implement best practices triggers a series of issues related to coding organization. Software development teams have to achieve business deadlines, so they adopt the bad practice to copy-and-paste code. In this way, clones populate software repositories and hinder the improvement or maintenance of systems. The most part of legacy systems code is the result of reusing existing code, so, developers who want to implement a new feature find some code snippet similar to the desired one then make a copy and modify it. A clone also result from identical instructions that works only with different data types – this indicate the failure to use Abstract Data Types. [1].

Construction of device drivers also generates many similarities between codes, as much of this type of program geared to the same platform is virtually identical [2]. Moreover, another reason for the existence of clones is called “reinventing the wheel”, because some developers do not bother to look if there is already a piece of code for something that was requested to the team [3].

The copy-and-paste way may cause a serious maintenance problem, for example, in case of bugs. If a bug was found in a piece of code that has been cloned in several other pieces, all of these clones should be corrected too [4].

Most studies around clone code theme make use of the same concepts. For example, the main types of code clones are [8]: Exact clones or program fragments identical to each other; Parameterized clones, are fragments with the same structure except for changes in data types, identifiers, layout and comments; Near-miss clones, program fragments copied with a few modifications inside; Semantic clones, blocks of code textually different but producing a same computation.

Other authors bring some terminologies concerning the relationships between clones [22]. A Clone Pair is a pair of code fragments identical or similar to each other. A Clone Class is a set of code fragments in which any two of the members can form a clone pair. In short, a clone class is the union of all clone pairs who shares code fragments in common. Clone Family, also known as Super Clone, is the group of all clone classes belonging to the same domain.

Despite code clones are considered harmful [15], for all the reasons we presented earlier, in some cases they may be a great deal. Introducing a new feature inside existing software can be eased by replicating the code and making the modifications. When the modified version of the code is tested in a sandbox or something similar, it can be applied in the production environment. This way minimizes the risk of instabilities in the stable version.

Some studies suggest that code clones may be avoided by adopting good design techniques and development methodologies, including refactoring on the development process [22]. Many efforts [9,11,12,13,14,16] show that code refactoring as part of the package of a clone detection tool may be a desirable feature in some situations.

Thus, considering a corporate environment with a well-defined software process, this paper aims address the following research question: "*Have corporate software environments lower clone incidence than Open Source Systems?* To answer, our experimental evaluation analyzed large-scale Closed-Source Systems and compared with OSS Systems published. The results showed similar numbers for both.

The remainder of this paper is structured as follows: Section 2 discusses related work. Section 3 is dedicated to the understanding of the main tool used in our experiment. Section 4 presents the experiment planning and definition. Section 5 describes the experiment execution among with the environment used to explore the clone code detection. Section 6 describes, analyzes and discusses the validity of the obtained results. Finally, Section 7 contains conclusions and final remarks.

**Related Works**

As this work focuses on clone incidence inside software repositories, this section presents studies about this subject. The main peculiarity of these articles regarding our work is that they were performed inside Open Source Software (OSS) environments.

Many works [4,10,11] were concerned about evaluating source code mining techniques and tools, identifying their strengths and weakness. Khatoon, Mahmood and Li [4] try to extract positive and negative aspects from cloned detection tools and techniques to help future researchers and developers.

Schwarz, Lungu and Robbes [7] focused on large code bases, combining three lightweight clone detection techniques to evaluate performance on a real-world ecosystem. The techniques are directed to three types of clones. The type 1 are Hashes of Source Code. Type 2 are defined as Hashes of Source Code With Renames. A clone is considered a type-2 if it is a type-1 even after every sequence of alphabetical letter be replaced by the letter "t" and all sequence of digits replaced by number 1. Type 3 or "Shingles", are defined as a consecutive sequence of tokens in a document, after the transformations defined by rules of type-2 clones.

Roy and Cordy [5] motivate our work. Like them, we run the NICAD tool in a software repository with the difference that is a

protected repository belonging to a private corporation. Their study was about finding function clones inside C and Java systems code repository, with projects varying in size from 4K LOC to 6265K LOC. All non-empty functions with a minimum of 3 LOC were considered, that includes the function header with opening and ending bracket and at least one code line. They used similarity thresholds, also known as Unique Percentage of Items (UPI) thresholds to present their results. The validation of NICAD results was done by hand and using Linux diff tool to check the textual similarities. The same researchers provided in another work [9] a description of commonly used terms, review of existing clone taxonomies, detection approaches and experimental evaluations of clone detection tools. At last, a list of some problems related to clone detection for future research is presented and discussed.

Clone Mining research needs substantial infrastructure support, particularly with respect to adopting a standard experimental process, described in some Mining Software papers [18] and in this paper, with the goal of effectively replicating clone studies. The barrier and cost for experimentation with Clones Mining are considerably low compared to other software engineering techniques (e.g., on-line experiments with participants). In other words, research projects and papers can conceive an experience factory and demonstrate true value of this area for practitioners.

**NICAD Tool**

NICAD Clone Detector [6] is an Open Source implementation of the NICAD method. It supports five languages, C, C#, Java, Python and WSDL, thru two granularity levels: blocks or functions. It is possible extend language support by adding a new TXL parser or extractor for the new language or granularity. This is possible though plugin architecture that also allows to add custom normalization templates. The TXL [16] is a hybrid functional/rule-base programming language, designed to support computer software analysis and source transformation tasks.

The NICAD method involves three stages [6]. The first one called Parsing, is about extract fragments of a given granularity, like functions or blocks to export them to a pretty-printed textual form, normalizing spacing and line breaks and removing comments to expose

identical clones as textually identical fragments. Normalization is the second stage that concerns to normalize, filter or abstract the extracted fragments before comparison. They can be renamed to adopt some standard or have some declarations removed. Lastly, comparison concerns to compare each fragment line using a LCS (longest common subsequence) algorithm to detect the clones. It is possible to parametrize NICAD with four different similarity thresholds, from exact to near-miss clones.

**Experiment**

Our work is presented here as an experimental process. It follows the guidelines by Wohlin et al. in [17]. In this section, we start introducing the experiment definition and planning. The following sections, will direct to the experiment execution and data analysis.

**Goal Definition**

Our goal is to compare clone findings of a private source code repository with another work that evaluates an Open Source software repository, using the same similarity thresholds.

To achieve this, we are going to execute an experiment in a controlled environment using the same tools, concepts and metrics than our main related work. This comparison test attempts to answer questions about clone incidence related to code freedom paradigms.

The goal is formalized using the GQM Goal template proposed by Basili and presented in [20]:

- **Analyze** our corporate projects
- **with the purpose of** evaluation (against published OSS projects)
- **with respect to** code clone manifestation
- **from the point of view of** the programmers
- **in the context of** software repositories

**Planning**

**Context selection.** The experiment will be off-line and executed with the NICAD clone detector inside a Java code repository containing about seven different systems of a corporate environment. The selected

subject organization is an educational-purpose company active in market since the 60s, with more than 2,000 employees and around 50,000 customers.

**Hypothesis formulation.** The research question for this experiment is: have corporate software environments lower clone incidence than Open Source Systems?

We will compare some extracted statistics of Java systems from open-source projects reported in Roy and Cordy [5] work with seven other private systems from our target corporation using the same extraction tool, respecting similarity thresholds between comparisons. To assure the reliability of our hypothesis test, we will calculate the average between the proportional results of exact similarity for each system (*S)*, where UPI threshold is 0.0. The proportion (*P)* is calculated by dividing Clone Pairs or Clone Classes Findings *(C)* by its respective KLOC.

When defining the variables for the formal test, the systems size was considered, because just the clone numbers does not imply conditions to evaluate a greater propensity to lower abstraction. Besides, the UPI threshold as 0.0 indicates an identical clone, evidencing more reliably the possibility of a type of Technical Debt (DT) [19] such as failure to code reuse or failure to use Abstract Data Types (ADT). Capture of Clone Classes were included in our experiment in order to identify repositories storing methods that are cloned in excess. Keeping this idea, we will try to confirm the following hypothesis:

*HYPOTHESIS 1.*

**Null hypothesis $H_0^{CP}$:** Source Code Repositories in the context of our corporate projects (1) have same incidence of clone pairs of the Open Source Projects (2) reported in the literature.

– $H_0^{CP}$: $\mu 1_{(\text{Clone Pairs Proportion})} = \mu 2_{(\text{Clone Pairs Proportion})}$

**Alternative hypothesis $H_1^{CP}$:** Source Code Repositories in the context of our corporate projects (1) have lower incidence of clone pairs than the Open Source Projects (2) reported in the literature.

– $H_1^{CP}$: $\mu 1_{(\text{Clone Pairs Proportion})} \neq \mu 2_{(\text{Clone Pairs Proportion})}$

*HYPOTHESIS 2.*

**Null hypothesis $H_0^{CC}$:** Source Code Repositories in the context of our corporate projects (1) have same incidence of clone classes of the Open Source Projects (2) reported in the literature.

– $H_0^{CC}$: $\mu 1_{(\text{Clone Classes Proportion})} = \mu 2_{(\text{Clone Classes Proportion})}$

**Alternative hypothesis $H_1^{CP}$:** Source Code Repositories in the context of our corporate projects (1) have higher incidence of clone classes than the Open Source Projects (2) reported in the literature.

– $H_1^{CC}$: $\mu 1_{(\text{Clone Classes Proportion})} \neq \mu 2_{(\text{Clone Classes Proportion})}$

**Independent variables.** NICAD method; Java OSS Projects reported in the literature; Our Java Industrial Projects. Moreover, the metrics used to evaluate this experiment were reused from the previously mentioned study [5]. We have used four UPI thresholds for the whole work: 0.0, 0.1, 0.2 and 0.3.

**Dependent variables.** The proportions and averages between results of clone pairs and classes and their respective KLOCs will be used as dependent variables. They are described as follows:

$$\textbf{Proportion}: P_S = C_S / KLOC_S$$
$$\textbf{Final Average}: \mu = (P_{S1} + P_{S2} + \ldots + P_{Sn}) / n$$

**Objects selection.** The private code projects size varied from a 6K LOC to a 35K LOC application. This selection was done by convenience. We have used some corporate projects which we were clone consultants for. The analysis is non-intrusive to developers as the data were drawn directly from the code repository, they did not know which source code would be extracted. Our repository contained only Java systems, to compare with the results of Open Source Java Systems obtained from the previously referenced study.

**Instrumentation.** We have used NICAD tool described in section 3. Results are printed to the standard output. Additional information results are exported to XML and HTML files in the same directory of the original system source.

**Experiment Operation**

In this section, we describe the whole experiment execution. The detection tool was configured to consider only functions or methods

with a minimum of 3 LOC. We do not analyze in this work clone distribution and localization over files or directories.

**Execution**

First, we extracted clone information for the whole repository to compare with the Open Source results using the Percentage of Total Cloned Methods (TCMp). Then, each project was analyzed individually and every clone-related discovered information was recorded and analyzed.

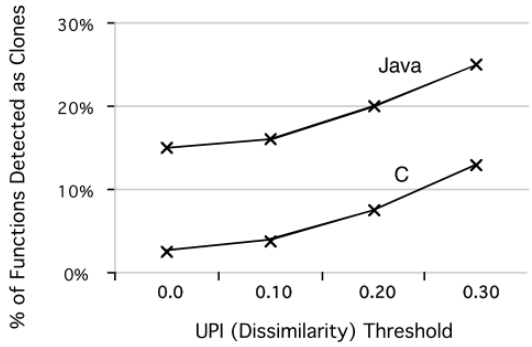


**Fig. 1.** TCMp values extracted from an OSS experiment [5].

At first, we can confirm that inside our Java repository there are fewer clone manifestations than the Open Source Java repository studied in Roy and Cordy work [5]. We may see in Figure 1 an incidence from 15% to almost 25% of TCMp, meanwhile in our experiment we did not reach 15% even using the highest UPI threshold value.

Our results also show that going from UPI value 0.0 to 0.3, the TCMp metric do not grow as the Java Open Source code. They show a 10% growth in clone appearance while in our repository clones increases only by 4% with the same test. The next step is to evaluate all projects individually.

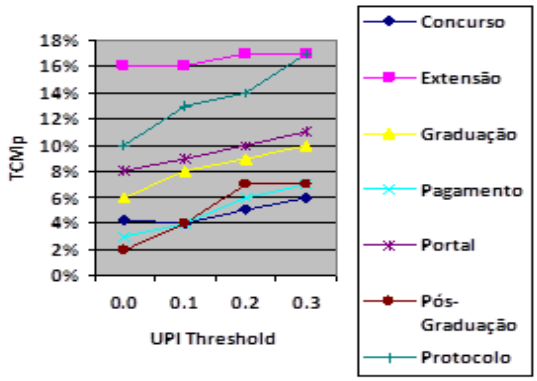


**Fig. 2.** TCM Percentage for Individual Systems

Running the NICAD clone detection tool for each system, we identified the projects with more and few clone incidences. We present these results in Figure 2.

Analyzing the graph, we note that clone incidence is not related to the project size. The bigger the worst does not apply here, since we have Graduação with less than 10% TCMp inside 35K LOC versus Extensão presenting more than 16% TCMp for only 7K LOC.

**Data Validation**

The NICAD clone detection tool generated HTML reports where we extracted the cloned methods to validate by hand the clone pairs with low similarity. Also, using Linux diff tool we compared the XML output file generated by NICAD to determine textual similarities of clones with the original source. This two-step process was the same used by the referenced work.

To ensure analysis, interpretation and validation, we used two types of statistical tests: Shapiro-Wilk Test and the Mann-Whitney test. Shapiro-Wilk test was used to verify normality of the samples. The Mann-Whitney test was used to check our hypothesis. All statistical tests were performed using the SPSS tool [21].

**Results**

To answer our experiment question, we executed all individual tests and created a table showing data to compare with the results

obtained from OSS experiment. The Table 1 shows several statistics collected from the analysis of our experiment.

**Analysis and interpretation**

Clone detection statistics from all the Java OSS analyzed are also present on Table 1. The "T" values on "Clone Pairs" and "Clone Classes" are representing the UPI thresholds. The OSS represent significantly much more LOC than corporate systems. The "PROP." column represents the proportional values for T=0.0. Ant project showed excellent results in comparison to the Academic System, both having about 35 KLOC. Ant returned less CPs and more CCs, which indicates a much lower clone incidence with the usage of methods abstraction. The worst performance was with Swing where proportionally presented more exact clones than Academic System.

For the OSS, Spule, JHotDraw and Jdtcore were the more clone-free projects. Analysis of Spule returned only 4,62% of clones. Following the same KLOC value, we can exemplify with the Protocol System, which we found 12,93% Clone Pairs. The corporate system with less Clone Pairs was the Payment System, with among 2% of exact clones. Post-grad System holds better performance about Clone Pairs, with 0,38% of incidence. The only OSS with clone incidence below 1% was Spule. The final average found for the OSS and Corporate Clone Pairs and Classes can be found on Table 2.

**Table 1.** Nicad statistics for OSS and Private code Repositories.

| REPOSITORY | PROJECT NAME | KLOC | CLONE PAIRS T= | | | | | CLONE CLASSES T= | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | 0.0 | $P_S$ (0.0) | 0.1 | 0.2 | 0.3 | 0.0 | $P_S$ (0.0) | 0.1 | 0.2 | 0.3 |
| Open Source Software | Ant | 35 | 363 | 10,37 | 365 | 374 | 426 | 92 | 2,63 | 94 | 101 | 119 |
| | EIRC | 11 | 117 | 10,64 | 117 | 121 | 149 | 35 | 3,18 | 35 | 36 | 47 |
| | Spule | 13 | 60 | 4,62 | 64 | 68 | 113 | 11 | 0,85 | 13 | 15 | 19 |
| | Javadoc | 14 | 193 | 13,79 | 197 | 240 | 304 | 80 | 5,71 | 82 | 95 | 110 |
| | JHotDraw | 40 | 291 | 7,28 | 295 | 377 | 598 | 137 | 3,43 | 141 | 170 | 208 |
| | Jdtcore | 148 | 1427 | 9,64 | 1553 | 2126 | 4378 | 323 | 2,18 | 377 | 518 | 660 |
| | Swing | 204 | 8115 | 39,78 | 8203 | 9978 | 11209 | 516 | 2,53 | 558 | 687 | 843 |
| Corporate Software | Concurso (Contest System) | 6 | 31 | 5,17 | 33 | 38 | 40 | 10 | 1,67 | 10 | 13 | 15 |
| | Extensão (Extension System) | 7 | 120 | 17,14 | 120 | 121 | 127 | 53 | 7,57 | 53 | 54 | 55 |
| | Pagamento (Payment System) | 8 | 16 | 2 | 20 | 47 | 89 | 5 | 0,63 | 7 | 10 | 11 |
| | Pós-Graduação (Post-grad System) | 8 | 30 | 3,75 | 32 | 38 | 40 | 3 | 0,38 | 5 | 9 | 9 |
| | Portal (Web Portal System) | 9 | 194 | 21,56 | 205 | 213 | 258 | 19 | 2,11 | 21 | 24 | 25 |
| | Protocolo (Protocol System) | 14 | 181 | 12,93 | 217 | 243 | 266 | 31 | 2,21 | 38 | 43 | 51 |
| | Graduação (Academic System) | 35 | 1280 | 36,57 | 1315 | 1401 | 1460 | 44 | 1,26 | 60 | 70 | 78 |

**Table 2.** Final average results

| FINDINGS | REPOSITORY | |
|---|---|---|
| | OSS | CORPORATE |
| CLONE PAIRS | 13,73 | 14,16 |
| CLONE CLASSES | 2,93 | 2,26 |

Based on these results, we observe that was not significant difference between the two kinds of repository. The Corporate Systems showed a little more proportional clone incidence than OSS. With this data, is not possible yet to make any assumption about results without sufficiently conclusive statistical evidence.

Firstly, we applied the Shapiro-Wilk test with a significance level of 0.05, analyzing the distribution normalization. The Sig variables (also known as p-values) for Clone Pairs were 0,003 on OSS samples and 0,370 on Industry samples. For Clone Classes, the p-values were 0,006 on Industry Systems and 0,541 on OSS. The numbers of at least one sample for each hypothesis were below the significance level, so, we assume that data distribution is not normal for all samples.

Applying the Mann-Whitney non-parametric test, we obtained a Sig. result of 0,110 for Clone Classes samples and 0,949 for Clone Pairs, both above the significance level of 0.05. Thus, the final decision is do not reject the null hypothesis. In fact, for Clone Pairs there was a strong retention for the null hypothesis $H0^{CP}$ ($\mu$1(Clone Pairs Proportion) = $\mu$2(Clone Pairs Proportion)). In real terms, there is a probability of almost 95% that we will mistakenly reject the similar clone incidence, although closed-source coding is easy to control, has stricter methodologies and more controlled development teams.

Systems with high incidence of clone pairs and much lower incidence of clone classes at the same time as the Academic System have methods that are excessively replicated. This represents a lot of code reuse and lack of using abstraction.

From the data extracted by NICAD, we calculated TCMp values for all projects. For each UPI threshold, we obtained the number of Clone Classes (CC). The CC were divided by the Potential Clone (PC) methods to obtain finally the percentage of TCM metric. The table also shows a Potential Clone Lines statistic capture by the tool, already filtered in pretty-printed format.

**Threats to validity**

In spite of the fact that our corporate systems are a mature, real world, large projects, and our results seem to be quite consistent with the systems sizes, our study shows threats to its validity that we must consider:

- We cannot conclude that all closed-source projects will present similar results as ours. Process maturity can play a large role on code clone manifestation;
- Other software characteristics such as complexity and programming paradigm may affect the results. We have not test for those variables;
- Adoption of design patterns also may influence on code clone manifestation;
- The profile of the development team (team size, age, experience) also can represent a change on the final sample.

**Conclusions and Future Work**

We found evidences in our experiment that clone incidence is not directly related to the size of code. In fact, the studied corporate systems had similar cloning incidence as the OSS ones. We encourage more research inside private environments to test hypothesis only studied on Open Source Software systems. In addition, our corporate systems had very few KLOCs than the Open Source ones. It is important to replicate this experiment inside several other private repositories to check if they present the same behavior. The more the systems are tested more we assure external validity.

As mentioned before, we adapted the software engineering experimental process described in Wohlin et al [12] to clones mining experiments. We believe that the studies, applications, and tools for software clone mining can benefit from this type of approach. Rigorous experimental description facilitates replication of studies and the executing of systematic reviews and other types of secondary analysis.

As future work, we have in mind a few projects related to clone incidence. The first one is to verify if the human profile of development team has some direct effect on clone appearance. Data like age, experience and qualification may be extracted and combined from

several sources to mount this profile. Other insight is to explore code comments to find out words that indicate something that was purposely implemented missing some pieces (for many reasons) and this will have to be done some time, indicating a Technical Debt (TD) issue.

**Acknowledgments**

This study could only be developed due to the support of Tiradentes University – UNIT, along with the Technology Information Department – DTI, who provided the repository used in our experiment.

**References**

[1] I. D. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier, "Clone detection using abstract syntax trees," in ICSM, 1998,pp. 368–377.

[2] Y. Ma and D. Woo. Applying a Code Clone Detection Method to Domain Analysis of Device Drivers. In Proceedings of the 14th Asia Pacific Software Engineering Conference (APSEC'07), pp. 254–261, Nagoya, Japan, December 2006.

[3] A. Marcus and J. I. Maletic. Identification of high-level concept clones in source code. 16th IEEE Intern. Conf. on Auto. Soft. Eng, pp. 107-114, San Diego, CA, USA, November 2001.

[4] S. Khatoon, A. Mahmood, and G. Li, "An evaluation of source code mining techniques," Proc. - 8th Int. Conf. Fuzzy Syst. Knowl. Discov. FSKD 2011, vol. 3, pp. 1929–1933,2011.

[5] C. K. Roy and J. R. Cordy, "An Empirical Study of Function Clones in Open Source Software," *15th Work. Conf. Reverse Eng.*, pp. 81–90,2008.

[6] J. R. Cordy and C. K. Roy, "The NiCad Clone Detector," *2011 IEEE 19th Int. Conf. Progr. Compr.*, no. Figure 3, pp. 219–220, 2011.

[7] N. Schwarz, M. Lungu, and R. Robbes, "On how often code is cloned across repositories," *Proc. - Int. Conf. Softw. Eng.*, pp. 1289–1292, 2012.

[8] D. Rattan, R. Bhatia, and M. Singh, *Software clone detection: A systematic review*, vol. 55, no. 7. Elsevier B.V., 2013.

[9] M. Kim, V. Sazawal, and D. Notkin, “An empirical study of code clone genealogies,” ACM SIGSOFT Softw. Eng. Notes, vol. 30, p. 187, 2005.

[10] C. K. Roy, J. R. Cordy, and R. Koschke, “Comparison and evaluation of code clone detection techniques and tools: A qualitative approach,” Sci. Comput. Program., vol. 74, pp. 470–495, 2009.

[11] D. Rattan, R. Bhatia, and M. Singh, *Software clone detection: A systematic review*, vol. 55, no. 7. Elsevier B.V., 2013.

[12] S. Sarala and M. Deepika, “Unifying clone analysis and refactoring activity advancement towards C# applications,”*4th Int. Conf. Comput. Commun. Netw. Technol. ICCCNT*, 2013.

[13] E. Duala-Ekoko and M. P. R. P. Robillard, “CloneTracker: Tool Support for Code Clone Management,” *Icse’08 Proc. Thirtieth Int. Conf. Softw. Eng.*, pp. 843–846, 2008.

[14] E. Duala-Ekoko and M. P. Robillard, “Tracking code clones in evolving software,” *Proc. - Int. Conf. Softw. Eng.*, pp. 158–167, 2007.

[15] C. J. Kapser and M. W. Godfrey, “‘cloning considered harmful’ considered harmful: Patterns of cloning in software,” *Empir. Softw. Eng.*, vol. 13, pp. 645–692, 2008.

[16] The TXL Programming Language. Available: http://www.txl.ca.

[17] Wohlin, P. Runeson, M. Host, M. C. Ohlsson, B. Regnell, and A. Wesslén (2000). Experimentation in software engineering: an introduction. Kluwer Academic Publishers, ISBN: 0-7923-8682-5.

[18] M. Colaço, M. Mendonça, M. André, D. F. Farias, and P. Henrique, “A Neurolinguistic-based Methodology for Identifying OSS Developers Context-Specific Preferred Representational Systems,” *Context*, no. c, pp. 112–121, 2012.

[19] Y. Guo, C. Seaman, R. Gomes, A. Cavalcanti, G. Tonin, F. Q. B. Da Silva, A. L. M. Santos, and C. Siebra, “Tracking technical debt - An exploratory case study,” *IEEE Int. Conf. Softw. Maintenance, ICSM*, pp. 528–531, 2011.

[20] R. van Solingen and E. Berghout (1999). The Goal/Question/Metric Method: A practical guide for quality improvement of software development. McGraw-Hill.

[21] SPSS, IBM Software, http://goo.gl/eXfcT3

[22] C. K. Roy and J. R. Cordy, "A Survey on Software Clone Detection Research," *Queen's Sch. Comput. TR*, vol. 115, p. 115, 2007

## 4. Artigo 2 (ICEIS)

# Procedural x OO: A Corporative Experiment on Source Code Clone Mining

José Jorge Barreto Torres
Federal University of Sergipe – UFS
Sao Cristovao, Sergipe, Brazil.
jorgesamango@gmail.com

Methanias C. R. Junior
Federal University of Sergipe – UFS
Sao Cristovao, Sergipe, Brasil.
mjrse@hotmail.com

## ABSTRACT

Open Source Software (OSS) repositories are widely used to execute studies around code clone detection, mostly inside the public scenario. However, corporative code Repositories have their content restricted and protected from access by developers who are not part of the company. Besides, there are a lot of questions regarding paradigm efficiency and its relation to clone manifestation. This article presents an experiment performed on systems developed in a large private education company, to observe and compare the incidence of cloned code between Object Oriented and Procedural proprietary software, using an exact similarity threshold. The results indicate that Object Oriented Software wondrously showed higher cloned lines of code incidence and a similar use of abstraction (clone sets) for functions or methods.

## Keywords

*Proprietary Software; Mining Software Repositories; Clones; Experimental Software Engineering; Closed-source projects.*

## 1. INTRODUCTION

The demand for speeding up software development allied to the lack of patterns and the inexistence of internal policies to implement best practices triggers a series of issues related to coding organization. Software development teams have to achieve business deadlines, so

they adopt the bad practice to copy-and-paste code. In this way, clones populate software repositories and hinder the improvement or maintenance of systems.

There are some reasons for the existence of clones: The most part of legacy systems code is the result of reusing existing code, so, developers who want to implement a new feature find some code snippet similar to the desired one then make a copy and modify it; Some code fragments used on default messages are copied to maintain a standard coding style, also generating clone code; Similar computing instances or code that perform similar computing are often cloned, even without the act of copy-and-paste, because the operations are similar; Some clones result from identical instructions that works only with different data types – this indicate the failure to use Abstract Data Types; Systems that have time constraints and need frequent optimization updates to computing replications, especially when the compiler does not provide inline expressions insertion; Occasional code fragments that are accidentally identical – as applications increase in size this type of accident occurs more often [1].

Construction of device drivers also generates many similarities between codes, as much of this type of program geared to the same platform is virtually identical, only having some attributes and parameters modified [2]. Moreover, another reason for the existence of clones is called “reinventing the wheel”, because some developers do not bother to look if there is already a piece of code for something that was requested to the team [3].

Despite the copy-and-paste way be more productive, this attitude may cause a serious maintenance problem, for example, in case of bugs. If a bug was found in a piece of code that has been cloned in several other pieces, all of these clones should be corrected so that the bug is completely resolved [4].

Most studies around clone code theme make use of the same concepts. For example, the main types of code clones are [8]: Exact clones or program fragments identical to each other; Parameterized clones, are fragments with the same structure except for changes in data types, identifiers, layout and comments; Near-miss clones, program fragments copied with a few modifications inside; Semantic clones, blocks of code textually different but producing a same computation.

Other authors bring some terminologies concerning the relationships between clones [22]. A Clone Pair is a pair of code fragments identical or similar to each other. To illustrate, we can turn attention to Figure 1 and note that we have three code fragments which we will name in a short way as F1, F2 and F3. From these three fragments we can mount five clone pairs: <F1(a),F2(a)>, <F1(b),F2(b)>, <F2(b),F3(a)>, <F2(c),F3(b)> and finally <F1(b),F3(a)>. A Clone Class is a set of code fragments in which any two of the members can form a clone pair. In short, a clone class is the union of all clone pairs who shares code fragments in common. Clone Family, also known as Super Clone, is the group of all clone classes belonging to the same domain.

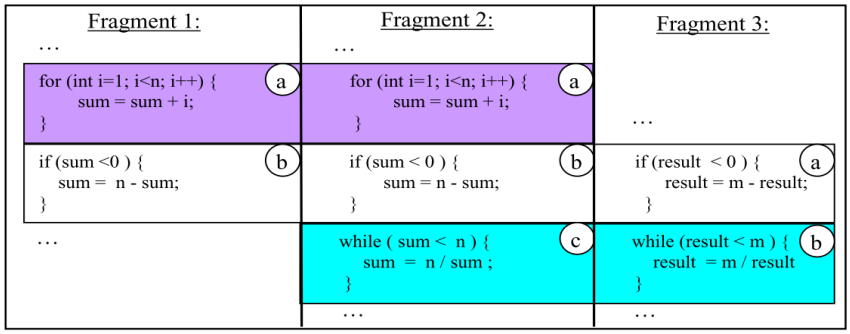


**Figure 1.** Examples of clone pair and class [22].

Despite code clones are considered harmful [15], for all the reasons we presented earlier, in some cases they may be a good choice. Introducing a new feature inside existing software can be eased by replicating the code and making the modifications. When the modified version of the code is tested in a sandbox or something similar, it can be applied in the production environment. This way minimizes the risk of instabilities in the stable version.

Some studies suggest that code clones may be avoided by adopting good design techniques and development methodologies, including refactoring on the development process [22]. Many efforts [9,11,12,13,14,16] show that code refactoring as part of the package of a clone detection tool may be a desirable feature in some situations. [5] studied cloning incidence in both C and Java Open Source Systems, executing a mixed experiment with different paradigms, showing interesting results regarding Clone Classes and Clone Sets incidence.

An Open Source System is publicly accessible and people can modify and share it. The software where only one person, team or organization who created it has access to modifications is called proprietary or closed source software [19].

Thus, considering a corporate environment with a well-defined software process, this paper aims address the following research question: "*Have object-oriented software systems more efficiency than procedural systems, regarding code clone manifestation?"* The question is about a proposal of efficiency in OO coding regarding Procedural, due to present abstraction structures in the Object-Oriented paradigm. The utilization of those abstraction structures are intended to provide a better code reuse and consequently less clone manifestation. To answer, our experimental evaluation analyzed large-scale Closed-Source Systems and compared their OO Systems with the Procedural ones. This is an *in-vivo* evaluation and the results are generalizable evidence only for similar teams, projects and environments. Despite OO languages are intended to have a better abstraction implementation, in our industrial environment, Procedural and OO systems presented similar behavior regarding clone manifestation. Those results showed numbers that leave opened other issues who are not directly linked to the paradigm question.

The remainder of this paper is structured as follows: Section 2 discusses related work. Section 3 is dedicated to the understanding of the main tool used in our experiment. Section 4 presents the experiment planning and definition. Section 5 describes the experiment execution among with the environment used to explore the clone code detection. Section 6 describes, analyzes and discusses the validity of the obtained results. Finally, Section 7 contains conclusions and final remarks.

## 2. RELATED WORKS

As this work focuses on clone incidence inside software repositories, this section presents studies about this subject. The main peculiarity of these articles regarding our work is that they were performed inside Open Source Software (OSS) environments.

Roy and Cordy [5] developed a qualitative evaluation along with a comparison of techniques and clone detection tools. In their work, some key concepts also have been described with a generic clone detection process and taxonomy. They used a hybrid clone detection tool called NICAD to examine more than 15 open source C and Java systems.

The same researchers mentioned above provided in another work [9] a description of commonly used terms, review of existing clone taxonomies, detection approaches and experimental evaluations of clone detection tools. At last, a list of some problems related to clone detection for future research is presented and discussed.

Many works [4,10,11] were concerned about evaluating source code mining techniques and tools, identifying their strengths and weakness. Khatoon, Mahmood and Li [4] try to extract positive and negative aspects from cloned detection tools and techniques to help future researchers and developers.

Schwarz, Lungu and Robbes [7] focused on large code bases, combining three lightweight clone detection techniques to evaluate performance on a real-world ecosystem. The techniques are directed to three types of clones. The type 1 are Hashes of Source Code. Type 2 are defined as Hashes of Source Code With Renames. A clone is considered a type-2 if it is a type-1 even after every sequence of alphabetical letter be replaced by the letter “t” and all sequence of digits replaced by number 1. Type 3 or “Shingles”, are defined as a consecutive sequence of tokens in a document, after the transformations defined by rules of type-2 clones.

Roy and Cordy [5] motivate our work. Their study was about finding function clones inside C and Java Open Source Code repositories, with projects varying in size from 4K LOC to 6265K LOC. All non-empty functions with a minimum of 3 LOC were considered, that includes the function header with opening and ending bracket and at least one code line. The validation of results was done by hand and using Linux *diff* tool to check the textual similarities. Like them, we run a clone detection tool in two different software repositories with the difference that they are protected repositories belonging to a private corporation. We intend to compare incidences of cloned code between Object-Oriented and Procedural Projects.

Clone Mining research needs substantial infrastructure support, particularly with respect to adopting a standard experimental process, described in some Mining Software papers [18] and in this paper, with the goal of effectively replicating clone studies. The barrier and cost for experimentation with Clones Mining are considerably low compared to other software engineering techniques (e.g., on-line experiments with participants). In other words, research projects and papers can conceive an experience factory and demonstrate true value of this area for practitioners.

## 3. CLONEDR TOOL

CloneDR uses a tree-based technique called Abstract Syntax Trees (AST). Supporting a variety of language dialects and the capacity of huge sets of files analysis, this tool is top-rated in literature [10] with a more sophisticated detection of clones. The intention for choosing this tool was to maintain the same type of analysis and results pattern of the referenced work. Also, the tool was the only one available to our team, for Java and PL/SQL code analysis.

The AST technique consists in receiving tree-parsed code fragments to find exact clones by hashing the sub-trees and comparing them. To locate near-miss clones, a bad-hashing function is used to preserve the main properties of this type of clone. For example, this function may ignore only identifier names, building a hash code for the rest.

A better description of this technique is presented by [1]. At first, all the program code is fragmented in parts that will be compared to find out which one are equivalent. After this parsing stage, an Abstract Syntax Tree is build and some algorithms are applied to find clones. The first one is called the Basic algorithm and it is responsible to detect sub-tree clones. The second one, called the sequence algorithm tries to detect variable-size sequences of sub-tree clones and it is used essentially to detect statement and declaration sequence clones. The third algorithm attempts to find more complex near-miss clones, generalizing combinations of other clones. AST technique does not concern to detect semantic clones. Some other semantic analysis technique may be used to capture different fragments but that produces similar results.

## 4. EXPERIMENT

Our work is presented here as an experimental process. It follows the guidelines by Wohlin et al. in [17]. In this section, we start introducing the experiment definition and planning. The following sections, will direct to the experiment execution and data analysis.

### 4.1. Goal definition

Our goal is to compare clone findings between two private source code repositories, one with Object-Oriented code and other with Procedural Projects, using an exact-similarity threshold.

To achieve this, we are going to execute an experiment in a controlled environment using a clone detection tool. This comparison test attempts to answer questions about clone incidence related to programming language paradigms.

The goal is formalized using the GQM Goal template proposed by Basili and presented in [20]:

- **Analyze** our corporate projects
- **with the purpose of** evaluation OO Systems against Procedurals Systems
- **with respect to** code clone manifestation
- **from the point of view of** the programmers
- **in the context of** an environment with a well-defined software process

### 4.2. Planning

**Context selection:** The experiment will be off-line and executed with the CloneDR clone detector inside a Java and a PL/SQL code repository containing about seven different systems each. The selected subject organization is an educational-purpose company active in market since the 60s, with more than 2,000 employees and around 50,000 customers. The PL/SQL development team differs from the java team by more experience and job constancy, as shown in Table 1. PL/SQL team consists of 4 developers with age from 37 to 40 years old and an experience average of 15 years against 5 developers for java team,

starting with 23 years old, most with only a 2-year programming experience. About 20 systems are maintained by procedural language team and 11 systems by OO team. Deadline pressure levels for both teams are the same.

**Table 1.** Experience and Constancy of Development Teams

| TEAM | DEVELOPER ID | AGE | YEARS OF LANGUAGE EXPERIENCE | YEARS ON COMPANY |
|---|---|---|---|---|
| PL/SQL TEAM | 1 | 40 | 18 | 20 |
| | 2 | 43 | 15 | 17 |
| | 3 | 38 | 13 | 15 |
| | 4 | 37 | 8 | 11 |
| JAVA TEAM | 5 | 43 | 13 | 16 |
| | 6 | 29 | 8 | 4 |
| | 7 | 27 | 7 | 2 |
| | 8 | 25 | 6 | 2 |
| | 9 | 23 | 5 | 2 |

**Hypothesis formulation:** The research question for this experiment is: Have object-oriented software systems more efficiency than procedural systems, regarding code clone manifestation?

Since private organizations provide a more controlled environment to adopt standardization of software development, we are interested about differences in the incidence of clones within programming language paradigm code repositories.

We will compare some extracted statistics of our Java systems with seven other PL/SQL private systems from our target corporation using the same extraction tool, respecting the similarity threshold between comparisons.

To assure the reliability of our hypothesis test, we will calculate the average between the proportional results of exact similarity for each system (*S)*, where similarity threshold is 1 (means 100% or exact clones). The proportion (*P)* is calculated by dividing Cloned Source Lines of Code or Clone Sets *(C)* by its respective total of Source Lines of Code (SLOC).

When defining the variables for the formal test, the systems size was considered, because just the clone numbers does not imply

conditions to evaluate a greater propensity to lower abstraction. Besides, the similarity threshold as 1 indicates an identical clone, evidencing more reliably the possibility of a type of Technical Debt (DT) [19] such as failure to code reuse or failure to use Abstract Data Types (ADT).

Capture of Clone Sets were included in our experiment in order to identify repositories storing methods that are cloned in excess.

Keeping this idea, we will try to reinforce the following hypothesis:

*HYPOTHESIS 1*

- **Null hypothesis $H0^{SL}$:** Object-Oriented Systems (1) have same incidence of Cloned SLOC than Procedural Systems (2) in the context of our corporate projects.
    - $H0^{SL}$: $\mu 1_{(\text{Cloned SLOC Proportion})} = \mu 2_{(\text{Cloned SLOC Proportion})}$
- **Alternative hypothesis $H1^{SL}$:** Object-Oriented Systems (1) have lower incidence of Cloned SLOC than the Procedural Systems (2) in the context of our corporate projects.
    - $H1^{SL}$: $\mu 1_{(\text{Cloned SLOC Proportion})} < \mu 2_{(\text{Cloned SLOC Proportion})}$

*HYPOTHESIS 2*

- **Null hypothesis $H0^{CS}$:** Object-Oriented Systems (1) have same incidence of Clone Sets than Procedural Systems (2) in the context of our corporate projects.
    - $H0^{CS}$: $\mu 1_{(\text{Clone Sets Proportion})} = \mu 2_{(\text{Clone Sets Proportion})}$
- **Alternative hypothesis $H1^{CS}$:** Object-Oriented Systems (1) have lower incidence of Clone Sets than the Procedural Systems (2) in the context of our corporate projects.
    - $H1^{CS}$: $\mu 1_{(\text{Clone Sets Proportion})} < \mu 2_{(\text{Clone Sets Proportion})}$

**Independent variables**: AST method; Our Object-Oriented and Procedural Industrial Projects, written respectively in Java and PL/SQL. Moreover, the parameters used to configure the tool used on this experiment will be described in Section 5.

**Dependent variables**: The Clone Sets and Cloned SLOC proportions ($P_S$) and averages ($\mu$) between results of Cloned SLOC and Clone Sets ($C_S$) and their respective SLOC will be used as dependent variables. They are described as follows:

- **Proportion**: $P_S = C_S/SLOC$
- **Final Average**: $\mu = (P_{S1} + P_{S2} + \ldots + P_{Sn}) / n$

**Objects selection:** The selection of Object-oriented and procedural projects is shown in Table 2, describing their names, amount of LOC and the kind of repository they belong to. The private code projects size varied from a 4.7K SLOC to a 102K SLOC application. This selection was done by convenience. We have used some corporate projects which we were clone consultants for. The analysis is non-intrusive to developers as the data were drawn directly from the code repository, they did not know which source code would be extracted.

**Instrumentation:** We have used CloneDR tool described in section 3. Results are printed to the standard output. Additional information results are exported to HTML files in the same directory of the original system source.

**Table 2.** Overview of selected projects

| REPOSITORY | PROJECT NAME | SLOC |
|---|---|---|
| OO | Graduação (Academic System) | 26462 |
| | Concurso (Contest System) | 6719 |
| | Extensão (Extension System) | 6693 |
| | Pagamento (Payment System) | 7743 |
| | Portal (Web Portal System) | 9648 |
| | Pós-Graduação (Post-grad System) | 4755 |
| | Protocolo (Protocol System) | 13739 |
| PROCEDURAL | Concurso (Contest System) | 14059 |
| | EAD (Distance Learning System) | 56406 |
| | Professor System | 18744 |
| | Protocolo (Protocol System) | 53298 |
| | Graduação (Academic System) | 102223 |
| | Financeiro (Finance System) | 68553 |
| | Pós-Graduação (Post-grad System) | 29735 |

## 5. EXPERIMENT OPERATION

In this section, we describe the whole experiment execution. The detection tool was configured to consider only functions or methods

with a minimum of 6 LOC. We do not analyze in this work clone distribution and localization over files or directories.

### 5.1. Execution

First, we extracted clone information for the whole OO repository to compare with the Procedural Repository results, using the CloneDR tool. Then, each project was analyzed individually still with the same tool and every clone-related discovered information was recorded and analyzed by hand.

At first we can confirm that inside our PL/SQL repository there are fewer clone manifestations than the Java repository. We may see a mean of proportional Cloned SLOC for the Procedural repository of 11,20%, meanwhile inside the OO repository the mean is 19,54% using the highest similarity threshold value.

### 5.2. Data validation

The CloneDR clone detection tool generated HTML reports where we extracted the cloned methods to validate by hand a sample of the cloned methods. This brings more confidence on what was analyzed by the clone detection tool.

To ensure analysis, interpretation and validation, we used two types of statistical tests: Shapiro-Wilk Test and the T-Test. Shapiro-Wilk test, normally applied on smaller populations, was used to verify normality of the samples. The T-Test was used to check our hypothesis. All statistical tests were performed using the SPSS tool [21].

## 6. RESULTS

To answer our experiment question, we executed all individual tests and created a table showing data to compare with the results obtained from the experiment. The Table 2 already showed several statistics collected from the analysis of our experiment.

### 6.1. Analysis and interpretation

Clone detection statistics from all the OO Projects analyzed are also present on Table 3.

The values on “Cloned SLOC” and “Clone Sets” are representing the results after an analysis using an exact-similarity threshold. The

Procedural projects presented significantly much more SLOC than OO systems. The "$P_S$" column represents the proportional values for the clone detection, for the respective system.

**Table 3.** Clonedr statistics for OO and Procedural code repositories

| PARADIGM | PROJECT NAME (*S*) | SLOC | CLONED SLOC | | CLONE SETS | |
|---|---|---|---|---|---|---|
| | | | $C_S$ | $P_S$ | $C_S$ | $P_S$ |
| Object-Oriented | Extensão (Extension System) | 6693 | 691 | **10.32** | 29 | **0.43** |
| | Concurso (Contest System) | 6719 | 902 | **13.42** | 31 | **0.46** |
| | Pós-Graduação (Post-grad System) | 4755 | 678 | **14.26** | 31 | **0.65** |
| | Pagamento (Payment System) | 7743 | 1706 | **22.03** | 51 | **0.66** |
| | Portal (Web Portal System) | 9648 | 2202 | **22.82** | 75 | **0.78** |
| | Graduação (Academic System) | 26462 | 6359 | **24.03** | 209 | **0.79** |
| | Protocolo (Protocol System) | 13739 | 4106 | **29.89** | 112 | **0.82** |
| Procedural | Concurso (Contest System) | 14059 | 966 | **6.87** | 76 | **0.54** |
| | Pós-Graduação (Post-grad System) | 29735 | 2506 | **8.43** | 169 | **0.57** |
| | EAD (Distance Learning System) | 56406 | 5002 | **8.87** | 335 | **0.59** |
| | Professor System | 18744 | 2150 | **11.47** | 142 | **0.76** |
| | Financeiro (Finance System) | 68553 | 9459 | **13.80** | 453 | **0.66** |
| | Graduação (Academic System) | 102223 | 14692 | **14.37** | 822 | **0.80** |
| | Protocolo (Protocol System) | 53298 | 7786 | **14.61** | 420 | **0.79** |

Analyzing the Table 3, we note that clone incidence is not related to the project size. The bigger the worst does not apply here, since we have the Java version of Academic System with 24% Cloned SLOC inside 26K SLOC versus the PL/SQL version presenting 14% Cloned SLOC for 102K SLOC.

PL/SQL Contest System showed excellent results in comparison to the OO Protocol System, both having about 14K SLOC. The OO Protocol System returned the higher Clone Set value, which indicates a worse use of methods abstraction. This system also had the worst performance, with almost 30% of cloned code.

Academic and Protocol System were the top-cloned software. Besides having a huge number of SLOC, they are maintained by a vast and heterogeneous development team.

For the OO, Extension System was the more clone-free project. The Procedural system with less proportionally Cloned SLOC was the Contest System, with among 7% of exact clones. The final average found for the OO and Procedural Cloned SLOC and Clone Sets can be found on Table 4.

**Table 4.** Final average results

| FINDINGS | PARADIGM | |
|---|---|---|
| | OO | PROCEDURAL |
| CLONED SLOC | 19.54 | 11.20 |
| CLONE SETS | 0.66 | 0.67 |

Based on these results, we observe that was some significant difference between the two kinds of repository. The Object-oriented Systems showed more proportional clone incidence than the Procedural ones. With this data, is not possible yet to make any assumption about results without sufficiently conclusive statistical evidence.

Firstly, we applied the Shapiro-Wilk test with a significance level of 0.05, analyzing the distribution normalization. The Sig variables (also known as p-values) for Cloned SLOC were 0.615 on OO samples and 0.261 on Procedural samples. For Clone Sets, the p-values were 0.216 on Procedural Systems and 0.193 on OO. The numbers on all samples for each hypothesis were above the significance level, so, we assume that data distribution is normal.

Applying the T-Test (Figure 2), we obtained a Sig. result of 0.014 for Cloned SLOC samples and 0.818 for Clone Sets. Only the p-value for Clone Sets was above the significance level of 0.05. This means that, regarding Cloned SLOC, we cannot assert the null hypothesis for $H_0^{SL}$. In other words, the differences of cloned single lines of code found on object-oriented programs was relatively higher than the numbers returned from procedural systems.

The Levene's Test is used to test if the samples have equal variances, also called homogeneity of variance. The sig value for CSETS is 0.466 (higher than 0.05) which means that, for Clone Sets the scores do not vary too much. Observing Source Lines of Code, the sig value is 0.021 (less than 0.05). Because of this, for SLOCS there is a statistically significant difference between the means.

**Independent Samples Test**

| | | Levene's Test for Equality of Variances | | t-test for Equality of Means | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | 95% Confidence Interval of the Difference | |
| | | F | Sig. | t | df | Sig. (2-tailed) | Mean Difference | Std. Error Difference | Lower | Upper |
| SLOCS | Equal variances assumed | 7,002 | ,021 | 2,867 | 12 | ,014 | 8,33571 | 2,90706 | 2,00177 | 14,66966 |
| | Equal variances not assumed | | | 2,867 | 8,360 | ,020 | 8,33571 | 2,90706 | 1,68190 | 14,98953 |
| CSETS | Equal variances assumed | ,566 | ,466 | -,236 | 12 | ,818 | -,01714 | ,07279 | -,17573 | ,14145 |
| | Equal variances not assumed | | | -,236 | 10,716 | ,818 | -,01714 | ,07279 | -,17787 | ,14358 |

**Figure 2.** T-Test results. Exported from IBM SPSS.

For the Clone Sets null hypothesis $H0^{CS}$, the final decision is to not reject it. In fact, for Clone Sets there was a strong retention for the null hypothesis $H0^{CS}$ (μ1(Clone Sets Proportion) = μ2(Clone Sets Proportion)). In real terms, there is a probability of almost 82% that we will mistakenly reject the similar Clone Sets incidence, although Object-Oriented coding has features that make easy code abstraction and reuse.

The results indicate that the use of abstraction for both Procedural and OO programs in this organization present a similar efficiency. We have more Cloned SLOC for OO than Procedural projects, but when implementing abstraction in functions or methods, the clone findings are almost equal. This means that, although object-oriented languages provide means to a better use of abstraction (e.g. polymorphism), the analyzed Java repository showed an inadequate behavior for this issue.

The development teams are different for PL/SQL and Java. The PL/SQL team has a characteristic of having lower staff turnover than Java team. Thus, the procedural repository takes advantage of owning maintainers with more experience time inside the company, with a solid knowledge about the business rules and knowing more deeply the code.

Moreover, is evident that, even with design patterns and frameworks adopted by the OO team, experience may have great influence on the capacity of abstraction. In background, there is a warning for the software management acting with regard to recycling and adoption of good practices by the teams.

For the organization, these results require further study about other causes that may have compromised the quality of coding. Features

concerning different development patterns or different team profiles as age, maturity and knowledge could be studied to check their interference on clones' manifestation.

### 6.2. Threats to Validity

In spite of the fact that our corporate systems are a mature, real world, large projects, and our results seem to be quite consistent with the systems sizes, our study shows threats to its validity that we must consider:

- We cannot conclude that all closed-source projects will present similar results as ours. Process maturity can play a large role on code clone manifestation;
- Other software characteristics such as complexity may affect the results. We have not test for those variables;
- Adoption of design patterns also may influence on code clone manifestation;
- The profile of the development team (team size, age, experience) also can represent a change on the final sample.

## 7. CONCLUSIONS AND FUTURE WORK

In our experiment, we analyzed two different repositories, which comprise systems of distinct programming language paradigms and found evidences that clone incidence is not directly related to the size of code. In fact, the studied Procedural systems had fewer lines of cloned code with much more coding lines than the OO ones.

The lack of code abstraction ended up being similar in both cases. Questions about the profile of both Java and PL/SQL development teams must be asked to check if experience, age, instruction degree and other factors, may affect the coding maintainability.

We encourage more research inside private environments to test hypothesis only studied on Open Source Software systems. Also, our corporate Object-Oriented Systems had very few SLOC than other Object-Oriented Open Source Systems. It is important to replicate this

experiment inside several other private repositories to check if they present the same behavior. The more the systems are tested, more we assure external validity.

As mentioned before, we adapted the software engineering experimental process described in Wohlin et al [12] to clones mining experiments. We believe that the studies, applications, and tools for software clone mining can benefit from this type of approach. Rigorous experimental description facilitates replication of studies and the executing of systematic reviews and other types of secondary analysis.

As future work, we have in mind a few projects related to clone incidence. The first one is to verify if the human profile of development team has some direct effect on clone appearance. Data like age, experience and qualification may be extracted and combined from several sources to mount this profile. Other insight is to explore code comments to find out words that indicate something that was purposely implemented missing some pieces (for many reasons) and this will have to be done some time, indicating a Technical Debt (TD) issue.

## 8. ACKNOWLEDGMENTS

This study could only be developed due to the support of Tiradentes University – UNIT, along with the Technology Information Department – DTI, who provided the repository used in our experiment.

## 9. REFERENCES

[1] I. D. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier, "Clone detection using abstract syntax trees," in ICSM, 1998,pp. 368–377.

[2] Y. Ma and D. Woo. Applying a Code Clone Detection Method to Domain Analysis of Device Drivers. In Proceedings of the 14th Asia Pacific Software Engineering Conference (APSEC'07), pp. 254–261, Nagoya, Japan, December 2006.

[3] A. Marcus and J. I. Maletic. Identification of high-level concept clones in source code.In Proceedings of the 16th IEEE International

Conference on Automated Software Engineering (ASE'01), pp. 107-114, San Diego, CA, USA, November 2001.

[4] S. Khatoon, A. Mahmood, and G. Li, "An evaluation of source code mining techniques," Proc. - 2011 8th Int. Conf. Fuzzy Syst. Knowl. Discov. FSKD 2011, vol. 3, pp. 1929–1933, 2011.

[5] C. K. Roy and J. R. Cordy, "An Empirical Study of Function Clones in Open Source Software," 2008 15th Work. Conf. Reverse Eng., pp. 81–90,2008.

[6] J. R. Cordy and C. K. Roy, "The NiCad Clone Detector," 2011 IEEE 19th Int. Conf. Progr. Compr., no. Figure 3, pp. 219–220, 2011.

[7] N. Schwarz, M. Lungu, and R. Robbes, "On how often code is cloned across repositories," Proc. - Int. Conf. Softw. Eng., pp. 1289–1292, 2012.

[8] D. Rattan, R. Bhatia, and M. Singh, Software clone detection: A systematic review, vol. 55, no. 7. Elsevier B.V., 2013.

[9] M. Kim, V. Sazawal, and D. Notkin, "An empirical study of code clone genealogies," ACM SIGSOFT Softw. Eng. Notes, vol. 30, p. 187, 2005.

[10] C. K. Roy, J. R. Cordy, and R. Koschke, "Comparison and evaluation of code clone detection techniques and tools: A qualitative approach," Sci. Comput. Program., vol. 74, pp. 470–495, 2009.

[11] D. Rattan, R. Bhatia, and M. Singh, Software clone detection: A systematic review, vol. 55, no. 7. Elsevier B.V., 2013.

[12] S. Sarala and M. Deepika, "Unifying clone analysis and refactoring activity advancement towards C# applications," 2013 4th Int. Conf. Comput. Commun. Netw. Technol. ICCCNT 2013, 2013.

[13] E. Duala-Ekoko and M. P. R. P. Robillard, "CloneTracker: Tool Support for Code Clone Management," Icse'08 Proc. Thirtieth Int. Conf. Softw. Eng., pp. 843–846, 2008.

[14] E. Duala-Ekoko and M. P. Robillard, “Tracking code clones in evolving software,” Proc. - Int. Conf. Softw. Eng., pp. 158–167, 2007.

[15] C. J. Kapser and M. W. Godfrey, “‘cloning considered harmful’ considered harmful: Patterns of cloning in software,” Empir. Softw. Eng., vol. 13, pp. 645–692, 2008.

[16] The TXL Programming Language. Available: http://www.txl.ca.

[17] Wohlin, P. Runeson, M. Host, M. C. Ohlsson, B. Regnell, and A. Wesslén (2000). Experimentation in software engineering: an introduction. Kluwer Academic Publishers, ISBN: 0-7923-8682-5.

[18] M. Colaço, M. Mendonça, M. André, D. F. Farias, and P. Henrique, “A Neurolinguistic-based Methodology for Identifying OSS Developers Context-Specific Preferred Representational Systems,” Context, no. c, pp. 112–121, 2012.

[19] OpenSource.com What is open source? Retrieved from https://opensource.com/resources/what-open-source

[19] Y. Guo, C. Seaman, R. Gomes, A. Cavalcanti, G. Tonin, F. Q. B. Da Silva, A. L. M. Santos, and C. Siebra, “Tracking technical debt - An exploratory case study,” IEEE Int. Conf. Softw. Maintenance, ICSM, pp. 528–531, 2011.

[20] R. van Solingen and E. Berghout (1999). The Goal/Question/Metric Method: A practical guide for quality improvement of software development. McGraw-Hill.

[21] SPSS, IBM Software, http://goo.gl/eXfcT3

[22] C. K. Roy and J. R. Cordy, “A Survey on Software Clone Detection Research,” Queen’s Sch. Comput. TR, vol. 115, p. 115, 2007.

## 5. Conclusão

O estudo da manifestação de clones em repositórios de código fonte é explorado por diversos pesquisadores que reúnem desde análise de algoritmos de detecção, passando por refatoração de código até revisões sistemáticas contendo comparativos de ferramentas e técnicas. A grande maioria dos experimentos é realizada em repositórios públicos, de código aberto, sem refletir a face industrial dessa temática.

Nesta dissertação, foram realizados experimentos com dados de dois tipos de repositórios, *Open e Closed-Source*. Como em grande parte da literatura, o primeiro experimento envolveu um repositório que abrigava códigos *Open Source,* mas, principalmente, um repositório privado. No segundo experimento, também em ambiente privado, foi comparada a incidência de clones entre repositórios contendo códigos de linguagens procedurais e orientadas a objetos. Nesses estudos experimentais, foi proposto o ataque às seguintes questões de pesquisa:

1. As linguagens com características do paradigma de orientação a objetos são mais eficientes do que as procedurais, com relação ao surgimento de clones?
2. Em repositórios de código aberto, existe uma tendência maior de aparecimento de clones?

A base industrial de código analisada foi fornecida por uma instituição de ensino superior, que mantém um vasto sistema acadêmico, além de outros produtos que assessoram a administração, tais como o ERP, o sistema contábil e o sistema de pagamento.

Tanto o primeiro quanto o segundo experimento apresentaram evidências de que a incidência de clones não estava diretamente ligada ao número de linhas de código que os sistemas possuíam. Após as devidas análises nos bancos de código-fonte industriais, foram obtidos resultados intrigantes: sistemas procedurais apresentaram menos incidência de clones que os orientados a objetos, reforçando a ideia de que linguagens com característica OO não necessariamente devem ser mais eficientes que as procedurais, no tratante ao surgimento de clones (questão de pesquisa 1).

Outra informação resultante dos nossos experimentos foi a conclusão que sistemas de código-fonte aberto mostraram quantidade similar de clones do que os sistemas proprietários abordados (questão de pesquisa 2). Nos dois trabalhos, foi sugerida a réplica dos experimentos em outros repositórios privados, para verificar se esse tipo de comportamento se repete, assegurando a validade dos nossos estudos.

Apesar dos nossos estudos terem se mostrado consistentes, devemos considerar algumas ameaças à validade desses resultados: maturidade dos processos, complexidade dos softwares, adoção de padrões de projetos e o perfil das equipes de desenvolvimento.

## 5.1. Contribuições

As contribuições obtidas a partir do desenvolvimento desse projeto são:

- Produção acadêmica apresentada em congresso de Qualis restrito (B1) e publicada em livro de editora internacional (Springer);
- Aplicação da detecção de clones em instituição privada de ensino superior, contribuindo para melhoria do código-fonte e consequentemente dos processos internos da equipe de desenvolvimento.

## 5.2. Trabalhos Futuros

Houve evidências que o perfil do desenvolvedor também pode ser algo que influencie diretamente o surgimento de clones de código e deve ser abordado em pesquisas futuras, respondendo à pergunta: “O perfil do desenvolvedor (idade, experiência) possui relação com a incidência de clones?”.

Além disso, considerando que um clone também é uma Dívida Técnica (KLINGER, 2011), uma outra questão de pesquisa pode ser investigada e respondida: ”Comentários de código podem ser usados para detecção da Dívida Técnica?”.

Além das questões sugeridas, o uso de padrões de projetos pode ser algo que também interfira na questão da manifestação de clones de código-fonte e seu estudo deve ser mais aprofundado.

# REFERÊNCIAS


ALVES, N. S. R.; RIBEIRO, L. F.; CAIRES, V.; MENDES, T. S.; SPINOLA, R. O. Towards an Ontology of Terms on Technical Debt. **2014 Sixth International Workshop on Managing Technical Debt**, p. 1–7, 2014. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6974882>. .

BAKER, B. On Finding Duplication and Near-Duplication in Large Software Systems.**Working Conference on Reverse Engineering.**1995, IEEE.

BALAZINSKA, M.; MERLO, E.; DAGENAIS, M.; LAGUE, B.; KONTOGIANNIS, K. Partial redesign of Java software systems based on clone analysis. **Sixth Working Conference on Reverse Engineering (Cat. No.PR00303)**, 1999.

BAXTER, I. D.; YAHIN, A.; MOURA, L.; et al. Clone Detection Using Abstract Syntax Trees. **International Conference on Software Maintenance, Proceedings**, p. 368–377, 1998.

CODABUX, Z.; WILLIAMS, B. Managing technical debt: An industrial case study. **2013 4th International Workshop on Managing Technical Debt, MTD 2013 - Proceedings**, p. 8–15, 2013.

CUNNINGHAM, W. The Wycash Portfolio Management System. **ACM SIGPLAN OOPS Messenger** (Vol. 4, No. 2). ACM. Dezembro 1992, pp. 29-30.

DUALA-EKOKO, E.; ROBILLARD, M. P. Tracking code clones in evolving software. **Proceedings - International Conference on Software Engineering**, p. 158–167, 2007.

GUO, Y. SEAMAN, C. GOMES, R. et al. Tracking technical debt - An exploratory case study. **IEEE International Conference on Software Maintenance, ICSM**, p. 528–531, 2011.

HIGO, Y.; KUSUMOTO, S. Code clone detection on specialized PDGs with heuristics. **Proceedings of the European Conference on Software Maintenance and Reengineering, CSMR**, p. 75–84, 2011.

HIGO, Y.; YASUSHI, U.; NISHINO, M.; KUSUMOTO, S. Incremental code clone detection: A PDG-based approach. **Proceedings - Working Conference on Reverse Engineering, WCRE**, p. 3–12, 2011.

ISHIHARA, T.; HOTTA, K.; HIGO, Y.; IGAKI, H.; KUSUMOTO, S. Inter-project functional clone detection toward building libraries - An empirical study on 13,000 projects. **Proceedings - Working Conference on Reverse Engineering, WCRE**, p. 387–391, 2012.

KAPSER, C. J.; GODFREY, M. W. Cloning considered harmful: Patterns of cloning in software. **Empirical Software Engineering**, v. 13, p. 645–692, 2008.

KHATOON, S.; MAHMOOD, A.; LI, G. An evaluation of source code mining techniques. **Proceedings - 2011 8th International Conference on Fuzzy Systems and Knowledge Discovery, FSKD 2011**, v. 3, p. 1929–1933, 2011.

KIM, M.; NOTKIN, D. Discovering and representing systematic code changes. **Proceedings - International Conference on Software Engineering**, p. 309–319, 2009.

KIM, M.; SAZAWAL, V.; NOTKIN, D. An empirical study of code clone genealogies. **ACM SIGSOFT Software Engineering Notes**, v. 30, p. 187, 2005.

KLINGER, Tim et al. An enterprise perspective on technical debt. In: **Proceedings of the 2nd Workshop on managing technical deb**t. ACM, 2011. p. 35-38.

LAGUE, B. PROULX, D. MERLO, E. MAYRAND J. HUDEPOHL, J. Assessing the Benefits of Incorporating Function Clone Detection in a Development Process. **International Conference on Software Maintenance.** 1997, IEEE.

MA, Y. S.; WOO, D. K. Applying a code clone detection method to domain analysis of device drivers. **Proceedings - Asia-Pacific Software Engineering Conference, APSEC**, p. 254–261, 2007.

MARCUS, A.; MALETIC, J. I. Identification of high-level concept clones in source code. **Proceedings 16th Annual International Conference on Automated Software Engineering (ASE 2001)**, , n. 0, p. 107–114, 2001.

RATTAN, D.; BHATIA, R.; SINGH, M. Software clone detection: A systematic review. [s.l.] **Elsevier B.V.**, 2013. v. 55

REHMAN, S. U.; KHAN, K.; FONG, S.; BIUK-AGHAI, R. An efficient new multi-language clone detection approach from large source code. **Conference Proceedings - IEEE International Conference on Systems, Man and Cybernetics**, p. 937–940, 2012.

ROY, C. K.; CORDY, J. R.; KOSCHKE, R. Comparison and evaluation of code clone detection techniques and tools: A qualitative approach. **Sci. Comput. Program.**, v. 74, p. 470–495, 2009.

SARALA, S.; DEEPIKA, M. Unifying clone analysis and refactoring activity advancement towards C# applications. **2013 4th International Conference on Computing, Communications and Networking Technologies, ICCCNT 2013**, 2013.

SCHWARZ, N.; LUNGU, M.; ROBBES, R. On how often code is cloned across repositories. **Proceedings - International Conference on Software Engineering**, p. 1289–1292, 2012.

SEVERINO, A. J. **Metodologia do trabalho científico**. 23. ed. [s.l.] Editora, Cortez, 2008.

TEMPERO, E. Towards a curated collection of code clones. **2013 7th International Workshop on Software Clones**, **IWSC 2013 -** Proceedings, p. 53–59, 2013.

TORRES, Jose J.; JUNIOR, Methanias C.; SANTOS, Francisco R. Mining Source Code Clones in a Corporate Environment. **In:**

**Information Technology: New Generations**. Springer International Publishing, 2016. p. 531-541.

WOHLIN, Claes et al. **Experimentation in software engineering**. Springer Science & Business Media, 2012.